AIAA-94-1297-CP

UM-PRS: AN IMPLEMENTATION OF THE PROCEDURAL REASONING SYSTEM FOR MULTIROBOT APPLICATIONS

Jaeho Lee, Marcus J. Huber, Edmund H. Durfee, Patrick G. Kenny

Artificial Intelligence Laboratory
The University of Michigan
Ann Arbor, Michigan 48109–2110

Abstract

The Procedural Reasoning System (PRS) is a general purpose reasoning system that is particularly suited for use in domains in which there are predetermined procedures for handling the situations that might arise. We have just completed an implementation of PRS written in C++, which we call the University of Michigan Procedural Reasoning System (UM-PRS). In this paper, we show how UM-PRS provides a critical level of representation for robotic applications in unpredictable domains, because it allows robotic vehicles to pursue long-term goals by adopting pieces of relevant procedures depending on the changing context, rather than having to blindly follow a prearranged plan. Specifically, UM-PRS has been used to control a real outdoor vehicle that changes its behavior based on what it sees in its environment. In turn, this provides the substrate for coordinating multiple robotic vehicles, allowing them to represent joint procedures and to infer each others plans through observation.

Introduction

We have been involved in a project which will, at its terminus, result in a team of robotic vehicles that are capable of autonomously working together while performing reconnaissance and other militarily relevant tasks. High–level plans for these vehicles will typically be in the form of mission plans and annotated maps. A mission plan is a declarative representation of the vehicles' major goals. Based on this, a human annotates a map showing topographical and strategic features to indicate where along planned routes some changes in each robotic vehicle's behaviors must be made (turned on, turned off, or parameters modified). Thus, the annotated map represents a "program" to be executed by the vehicles, where crossing certain spatial lines trigger a vehicle to execute the next step of its program.

The annotated map representation[8] is incomplete in that it lacks the richness required for robotic control in unpredictable, dynamic environments. Blindly following the preprogrammed sequence of behaviors might be hazardous (if the context in which the sequence was formed changes) or impossible (if the vehicle loses track of its position on the map or if its control is temporarily taken over by a human). Therefore, it is important that the map and mission plan be accompanied by more general knowledge about why certain actions are being taken and in what context. Military doctrine, as laid out in documents such as field manuals, contains large numbers of standard operating procedures (SOPs) that should be selectively invoked as conditions and objectives change. In fact, an annotated map for a mission plan typically represents a particular sequence of SOPs that are expected to be useful. For a flexible, autonomous system to succeed, however, the suite of SOPs must be available to the system at runtime.

The Procedural Reasoning System[4–6] is a general purpose reasoning system particularly suited for use in domains in which there are predetermined procedures for handling the situations that might arise. This makes it very applicable in domains such as that discussed above. However, until recently, there has not been an implementation of PRS that is freely available for use. We have just completed an implementation of PRS written in C++, which we call the University of Michigan Procedural Reasoning System (UM-PRS).

In this paper we discuss the details of our initial implementation. Our implementation is novel in terms of both knowledge representations and control structures all written in C++ to meet the needs of efficient real-time robotic control. First, we briefly introduce the general concepts of procedural reasoning systems, the specific representations and the interpreter of the initial UM-PRS version. Second, we illustrate how UM-PRS serves as an important intermediate level of representation and reasoning between the high-level mission plan and the executable annotated map, and how it can interface with these levels. We also illustrate how the flexibility provided by UM-PRS allows autonomous responses beyond those permitted by annotated maps alone. Third, we briefly describe how UM-PRS has been used, to date, in the dynamic control of a real outdoor vehicle and how UM-PRS provides a basis for multi-vehicle coordination, both in terms of allowing the automated formation of belief networks that a vehicle can use to infer the plans of others through observation, and in terms of representing the collective activities of vehicles and the

This research was sponsored in part by ARPA under contract DAAE-07-92-C-R012.

roles that they can play. Finally, we summarize the current status of UM-PRS and the ongoing improvements that we are making in order to realize the full, flexible autonomous capabilities in our multivehicle system.

Procedural Reasoning System

Developing reasoning systems that can reason and plan in continuously changing environments in real-time is emerging as an important area of application of Artificial Intelligence. In this section, we describe basic features of the Procedural Reasoning System (PRS) that motivated us to adopt PRS as a conceptual framework for our system.

The Procedural Reasoning System[4-6] is a general-purpose reasoning system, integrating traditional goal-directed reasoning and reactive behavior. Because most traditional deliberative planning systems formulate an entire course of action before starting execution of a plan, these systems are brittle to the extent that features of the world or consequences of actions might be uncertain. In contrast, the Procedural Reasoning System continuously tests its decisions (both high- and low-level) against its changing knowledge about the world, and can redirect the choices of actions dynamically while remaining purposeful to the extent of the unexpected changes to the environment.

PRS thus is not a planning system in the traditional AI sense, in that PRS does *not* concentrate on searching for sequences of primitive actions that lead to specific goals. Instead, PRS is a plan execution system: it assumes that it already has "plans" (procedures) for achieving various goals in various contexts; however, it might string together actions in unexpected ways as it dynamically chooses among procedures and subprocedures in a changing environment.

Typically, accomplishing a mission in a military setting is much more similar to the plan execution activities of PRS than to the activities of traditional planning systems. Procedures for military tasks such as reconnaissance are developed and learned off-line,[2] and training involves mastering the selection and execution of predefined procedures. These procedures may contain knowledge about both cognitive (such as situation assessment) and physical actions, and they can be arbitrarily complex. Often, a "step" in one procedure (such as "move to assembly area") might itself correspond to several sub-procedures, each appropriate in different contexts.

PRS is conceptually geared for representing precisely this kind of procedural information. Several features that make PRS particularly powerful as a situated reasoning system[6] are as follows:

- The semantics of its plan (procedure) representation, which is important for verification and maintenance.
- Its ability to expand and act on partial plans.
- Its ability to pursue goal-directed tasks while being responsive to changing patterns of events in bounded time.

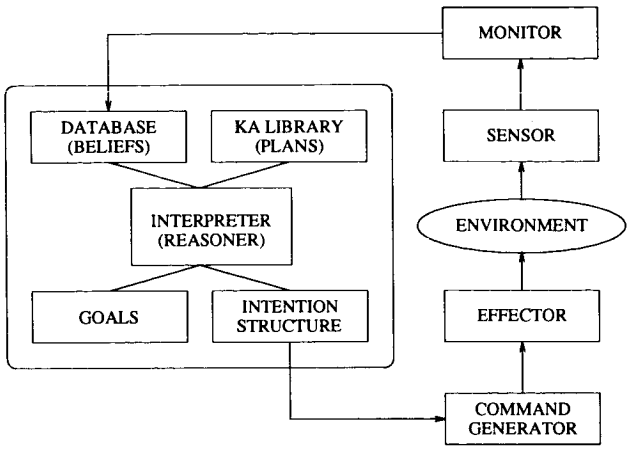

Figure 1: PRS System Structure[3]

- Its facilities for managing multiple tasks in real time.
- Its default mechanisms for handling the environment's stringent real-time demands.
- Its metalevel (or reflexive) reasoning capabilities.

PRS consists of (1) a *database* (called World Model) containing current *beliefs* or facts about the world; (2) a set of current *goals* to be realized; (3) a set of *plans* (called Knowledge Areas) describing how certain sequences of actions and tests may be performed to achieve given goals or to react to particular situations; and (4) an *intention structure* containing those plans that have been chosen for eventual execution (Figure 1). An *interpreter* (or *reasoning mechanism*) manipulates these components, selecting appropriate plans based on the system's beliefs and goals, placing those selected on the intention structure, and executing them.[3]

The system interacts with its environment, including other systems, through its database (which acquires new beliefs in response to changes in the environment) and through the actions that it performs as it carries out its intentions.[3]

UM-PRS

In our particular implementation of PRS, we have been concerned to a large extent with the specific requirements of the military task domain we have outlined, and which we will discuss further below. Thus, whereas some versions of PRS have been developed (typically in Lisp) to be extremely general, our goal has been to identify the crucial features of PRS for our application domain, and to implement them in a robust way in C++. Some of the design and implementation decisions that we have made follow.

World Model

In our implementation, WM is a database of facts which are represented as relations. A relation has a name and a variable number of fields. Initial facts are asserted at the beginning of a UM-PRS program by the user, and other facts can be either asserted or retracted by the KAs, which will be explained below.

Knowledge Areas

A Knowledge Area (KA) is a declarative procedure specification of how to satisfy a system goal or query. It consists of the *purpose* (a goal, query, test, or world model assertion or retraction) for executing the KA, the *context* in which the KA is applicable, a graphical network called the *body* which specifies what is required to satisfy the purpose in terms of primitive functions, subgoals, conditional branches, etc., and a symbol table which holds values for variables when a KA is instantiated for a specific situation. The context consists of a mixed sequence of patterns to be matched against the WM and expressions to be satisfied using the variable bindings generated during the matching.

A SOAK (Set Of Applicable KAs) is a collection of KAs which have been instantiated to achieve a goal (purpose) that has just been activated. Each KA in the SOAK is applicable to the specific situation, as one role of the context is to filter out KAs that are not relevant to a particular situation.

The body describes the procedure's steps, consisting of a network of actions. The body can be viewed as a plan schema. The schema is instantiated with the bindings which are generated when the purpose and the context of the KA are checked during SOAK generation.

Actions

Actions are the arcs in a KA body that constitute either primitive actions or subgoals to achieve. A "primitive" action is a behavior or activity that can be executed directly. Any other type of action represents a) a goal that needs achievement, maintenance, or to be waited upon b) a query, c) a test, or d) an assertion or retraction of world model information. Actions are represented by a base class that holds information regarding the KA in which the action is found and the action name. Each of the types of actions are represented by a derived class that maintains such information as function pointers (for a primitive action), the expression to evaluate (for a test), a goal or query expression, or a world model relation to assert or retract.

Goals

Goals in UM-PRS are the world states that the system is trying to bring about (or maintain, etc.). A goal can be either a top–level goal, which controls the system's highest order behavior, or a subgoal activated by the execution of a KA arc.

Intention Structure

The intention structure acts as the run–time stack for the system. It keeps track of the progress of each high–level goal and all of the subgoals. The intention structure suspends, resumes, cancels, and proceeds with execution of goals in much the same way as an operating system. The intention structure maintains information about what KAs are currently active, as well as what actions in each KA are to be executed next. As there are conditional branches in a KA, the intention structure must also maintain information regarding the success or failure of branches.

The Interpreter

The UM-PRS interpreter is similar to the interpreter described for PRS: It is what controls the execution of the entire system. Whenever there is new or changed information in the world model or goal list, the interpreter determines a new SOAK. From this SOAK is selected the most appropriate KA, which is placed in the intention structure. When there are no SOAKs being generated, the interpreter checks the intention structure for the currently active KAs and executes the next primitive action. If this action changes the goal list (by creating a subgoal or by satisfying a goal) or world model, a new SOAK is created and the cycle starts over. If a new SOAK is not created, then the next arc in a leaf-level KA is executed.

With this implementation, the interpreter facilitates switching to more important goals according to the situation. This implementation also stays committed to one method of achieving a goal by not reconsidering alternatives unless the current method fails. The UM-PRS interpreter is different from the PRS interpreter in that there is currently no metalevel control, although we plan on adding that in the near future as we enrich the set of KAs such that we could have many KAs applicable in overlapping situations.

Example

We began by briefly describing the incompleteness of the annotated map representation[8] in unpredictable, dynamic environments. In this section, we show examples of using both the annotated map representation and the UM-PRS system. We first show a clear correspondence between the annotated map representation and the UM-PRS representation when geographical events are the main triggers of the actions. In the second example, we show the limitation of the annotated map representation and, in contrast, the richness of UM-PRS when general (non-geographical) events trigger actions.

Figure 2 is an example annotated map representation of a simple scenario of getting to an observation point from the assembly area using road following and STRIPE (a waypoint following method) alternatively. The actions to be taken are annotated along the circles in the map. Figure 3 is an example UM-PRS knowledge area that generalizes the procedure on the annotated map. The actions in the KA body roughly correspond to the sequence of actions represented in the annotated map, and the KA representation (context and body) is somewhat simplified to highlight the correspondence between the two representations. A full detailed working example is presented in the Appendix.

An important advantage to using the procedural representation over the annotated map representation is that the correspondence between mission objectives and map markings is made explicit. That is, with an annotated map, the (human) mission planner has a

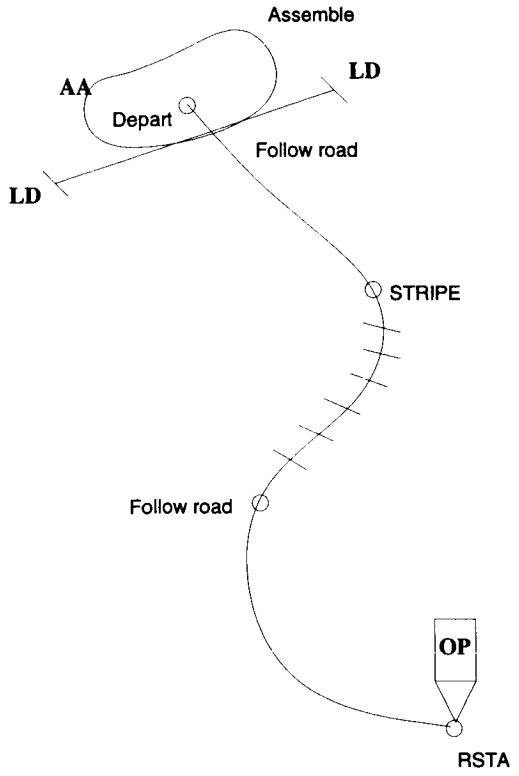

Figure 2: Annotated map

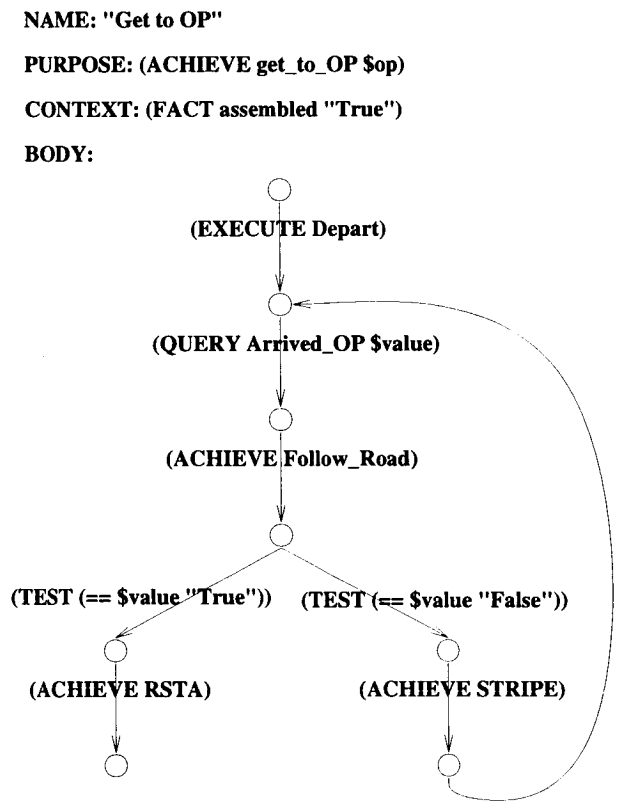

Figure 3: UM-PRS KA

mission in mind, along with an operating procedure to accomplish it. The overall operating procedure is not explicitly represented anywhere, but only the steps are given in the annotations. However, the human interface using a UM-PRS-based system can be quite different. The human can specify an objective, and UM-PRS will retrieve appropriate KAs. In order to instantiate those KAs, UM-PRS must bind variables to values, and those values could include geographical information (where the assembly area is, or which road to follow). Thus, while the user will certainly still point to locations and regions on a map, he or she will do so in response to the needs of the explicitly-represented, standard operating procedures currently being elaborated by UM-PRS.

The second example scenario is shown in Figure 4. The vehicle starts out by issuing a command to start the road-following behavior. This moves the vehicle forward until it reaches a cone or goes past a maximum allowed distance. If it passes the max distance without seeing the cone, the vehicle stops and the demo is done. If the vehicle detects the cone, it approaches the cone and starts off-road behavior until it sees that it has reached the end point. When the vehicle has reached the end point, the demo is done.

The idea of the demo is to show that UM-PRS can be used to represent conditional actions based on non-geographical events such as cone detection. The cone can be placed anywhere along the road, or there may be no cone at all. Such non-geographical events are hard to annotate in a map. The full KA description of this demo is presented in the Appendix. Note that the general capability of pattern-directed invocation of UM-PRS makes it possible to represent high-level choices of very different behaviors as well as simple non-geometric events. So, for example, representing procedures for context changes such as "running for cover" after having "been discovered by enemy" is easy to represent in UM-PRS, rather than cluttering up regions of the map with annotations.

Experiments

In this paper, we have described our implementation of UM-PRS. Currently, we are experimenting with using UM-PRS for robotic control of two indoor mobile robots, and also of an outdoor robotic vehicle (Figure 5). The scenario being used for our experiments is a reconnaissance task in a military domain. Typically, the means of satisfying the task's goals are described in terms of procedures to follow in specific situations. These procedures correspond exactly to KAs, with the context and the purpose specific to the situation in which it is applicable.

One thing we have to note is that all the actions described in the KA body should eventually map down to primitive actions (C or C++ functions) which are directly executable on the real robot or vehicle. Since our implementation is done in C++, the interface between KA actions and real primitive control functions is very natural and efficient.

UM-PRS: Toward Multi-vehicle Coordination

Many tasks of interest, particularly in the military domain, require the concerted efforts of several play-

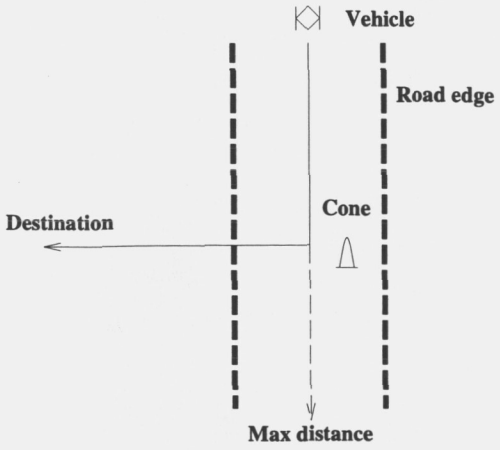

Figure 4: Cone Demo

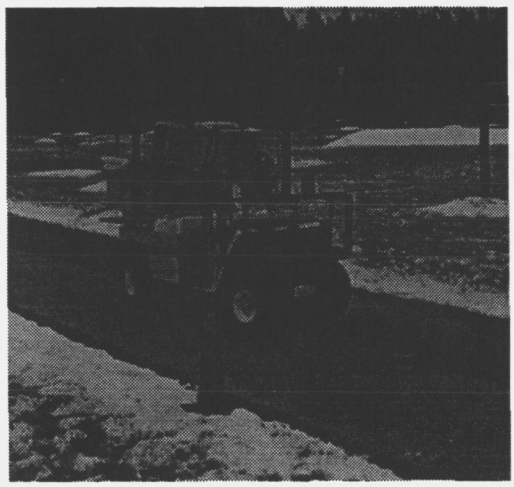

Figure 5: Vehicle and Cone

ers. In other words, they require teamwork. Recently, Georgeff's colleagues[7] have explored extensions to the procedural representation to model different "roles" played in team procedures, and have developed algorithms for assigning roles to potential team members. These capabilities need to be incorporated into UM-PRS to capture the notion of roles in military procedures, such as the roles of "bounder" and "overwatcher" in a bounding-overwatch procedure. Since, in such a procedure, the vehicles take turns watching and moving as they leapfrog across an area, the roles will be assigned and reassigned dynamically in the course of the procedure.

Having each adopted its role in a shared, team procedure, a vehicle can then work to achieve its goals in the procedure. However, since how it chooses to achieve its goals can impact the choices available to other vehicles in how they achieve their goals, some additional coordination is often needed. In essence, while the vehicles might commit to a team plan at an abstract level, they also might have to commit ahead of time to how they might (or might not) elaborate their plans into detailed actions. Anticipating and making necessary commitments ahead of time (before they move into the field where communication is more risky and error prone) should be done, but overcommitment should be avoided lest the vehicles commit to specific courses of action that they later find to be suboptimal or even ineffective. Thus, while the conceptual framework of PRS emphasizes delayed commitment to action until the action must be taken, coordination requires some degree of commitment to the future. Extending UM-PRS to provide this capability is one of our ongoing efforts.

Also, once in the field, some coordination might be needed, and is typically done through communicating about plans, goals, or beliefs about the world. Each of the involved vehicles can reason about this information in order to detect possible conflicts, improvements, synergies, etc. As mentioned before, explicit communication may not always be possible, however, due to such situations as hostile vehicles in the vicinity, environmental noise, and broken equipment. Plan recognition is the process of inferring the same information (the motivating goals or beliefs of a vehicle) based upon sensory observation of that vehicle's actions. Once the plans have been inferred, the same reasoning process can be used to make decisions regarding coordination.

One of the most significant issues that arises is that plans of the robotic vehicles will be in the form of UM-PRS Knowledge Areas, which are not conducive to performing plan recognition. In response to this, we are developing a system that will automatically convert the plans of the other team vehicles into a representation amenable to plan recognition, namely belief networks. Belief networks, also called Bayesian networks,[1] provide the framework and mechanisms for performing probabilistic reasoning about the relationship between the observations of a vehicle's actions and the vehicle's reasons (plans, goals, etc.) for performing that action. While manual construction of belief networks to represent the UM-PRS plans can be done, it is extremely time consuming and subject to variability. When considering the large number of possible KAs for complex tasks, and hence the large number of possible plans that might be created and executed, manual conversion of the plans becomes impractical. By providing an automated system, this process can be performed quickly, efficiently, and without variation.

Conclusions

The representation and control scheme that we have implemented has proven to be sufficiently powerful for planning and execution in procedurally rich domains, specifically in a robotic reconnaissance task. Our experiments have shown the initial implementation to reactively switch between goals, while remaining committed to the current method of achieving each goal. We are actively working on many extensions to the initial implementation (such as metalevel control, and coordination mechanisms, as outlined above) that will make it even more powerful and flexible.

Appendix: Example system

The example system included here as a demo for UM–PRS is an early version of a KA library and primitive functions developed to demonstrate the applicability of UM–PRS to mobile robot tasks.

The idea of the demo is to show that a planner can be used as a triggering device to enable and change vehicle behaviors.

- The vehicle starts out by issuing a command to start the YARF road following behavior. This moves the vehicle forward until it reaches a cone or goes past a maximum allowed distance.
- UM-PRS will wait until the vehicle is stopped (0 = not stopped, 1 = stopped). UM-PRS then will query to see if the vehicle has seen the cone or passed the max distance.
- If it passes the max distance, then the vehicle stops and the demo is done.
- If the vehicle detects the cone, then the vehicle stops and waits for the next behavior.
- If the vehicle sees the cone, then UM-PRS will issue an Approach Cone behavior. UM-PRS will then wait until the vehicle has stopped and check if it has reached the cone.
- If the vehicle has reached the cone, then UM-PRS will issue an Off Road behavior. UM-PRS will wait until the vehicle has stopped and reached the end point.
- When the vehicle has reached the end point, then the demo is done.

KA code

```
GOALS:
        (ACHIEVE cone_demo)

FACTS:
        (vehicle_status "True")
        (demo_done "False")
        (cone_found "False")
        (cone_reached "False")
        (vehicle_reached "False")
        (vehicle_maxdist "False")
        (vehicle_stopped "True")


//-------------------------------------
// KA 1
//-------------------------------------
KA {
NAME:
 "complete cone demo"

DOCUMENTATION:
 "This is the main KA
  that will start the cone demo"

PURPOSE:
 (ACHIEVE cone_demo)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT demo_done "False")

BODY:
 (1 (ACHIEVE vehicle_initialized) 2)
 (2 (ACHIEVE road_scouted) 3)
}

//-------------------------------------
// KA 2
//-------------------------------------

KA {
NAME:
 "initialized vehicle"

DOCUMENTATION:
 "initialized all the vehicle controls"

PURPOSE:
 (ACHIEVE vehicle_initialized)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT demo_done "False")

BODY:
 (99 (EXECUTE init_database 2) 1)
 (1  (EXECUTE home_robot $x $y $to) 2)
 (2  (ASSERT vehicle_initalized)  3)
 (3  (ASSERT YARF              2)  4)
 (4  (ASSERT APPROACHCONE      4)  5)
 (5  (ASSERT OFFROAD           8)  6)
 (6  (ASSERT CHECKVEHICLE     16)  7)
 (7  (ASSERT STOPPED           2)  8)
 (8  (ASSERT CONEFOUND         4)  9)
 (9  (ASSERT MAXDIST           8) 10)
 (10 (ASSERT READCHEDCONE     16) 11)
 (11 (ASSERT READCHEDVEHICLE 32) 12)
 (12 (ASSERT VEHICLESTATUS   64) 13)
}

//-------------------------------------
// KA 3
//-------------------------------------
KA {
NAME:
 "road scouted"

DOCUMENTATION:
 "This KA will scout out a road,
  as per the scenario plan"

PURPOSE:
 (ACHIEVE road_scouted)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT demo_done "False")

BODY:
 (1 (ACHIEVE road_followed_until_cone) 2)
 (OR
   ((2 (FACT cone_found $value) 3)
    (3 (TEST (== $value "True")) 5)
    (5 (ACHIEVE cone_approached) 6)
    (6 (FACT cone_reached $value) 7)
    (7 (TEST (== $value "True")) 8)
    (8 (ACHIEVE traveled_off_road) 9)
```

```
    (9 (FACT vehicle_reached $value) 10)
    (10 (TEST (== $value "True")) 11)
    (11 (ASSERT demo_done "True") 12))
   ((2 (FACT vehicle_maxdist $value) 20)
    (20 (TEST (== $value "True")) 21)
    (21 (ASSERT demo_done "True") 12)))
}

//-------------------------------------
// KA 4
//-------------------------------------
KA {
NAME:
 "road_followed_until_cone"

DOCUMENTATION:
 "This KA will follow the road
  until the vehicle sees a cone
  or passes the turn off point"

PURPOSE:
 (ACHIEVE road_followed_until_cone)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT cone_found "False")
 (FACT vehicle_maxdist "False")
 (FACT YARF $YARF)
 (FACT STOPPED $STOPPED)
 (FACT CONEFOUND $CONEFOUND)
 (FACT MAXDIST $MAXDIST)

BODY:
 (1 (EXECUTE start_behavior $YARF) 2)
 (2 (EXECUTE check_behavior
            $STOPPED
            $vehicle_stopped) 3)
 (3 (FACT vehicle_stopped $value) 4)
 (OR
   ((4 (TEST (== $value "False")) LOOP 2))
   ((4 (TEST (== $value "True")) 5)
    (5 (EXECUTE check_behavior
              $CONEFOUND
              $cone_found) 6)
    (6 (ASSERT  cone_found
              $cone_found) 7)
    (7 (EXECUTE check_behavior
              $MAXDIST
              $vehicle_maxdist) 8)
    (8 (ASSERT  vehicle_maxdist
              $vehicle_maxdist) 9)))

/*
  start YARF behavior
  while (not done)
   if (vehicle stopped)
         done = true
  if (vehicle max distance)
     stop all,
     we are done with demo,
     mission was not accomplished
  else if (cone found)
     assert found cone
*/
}

//-------------------------------------
// KA 5
//-------------------------------------
KA {
NAME:
 "cone_approached"

DOCUMENTATION:
 "When the vehicle sees the cone,
       it approaches it"

PURPOSE:
 (ACHIEVE cone_approached)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT cone_reached "False")
 (FACT APPROACHCONE $APPROACHCONE)
 (FACT STOPPED $STOPPED)
 (FACT REACHEDCONE $REACHEDCONE)

BODY:
 (1 (EXECUTE start_behavior
             $APPROACHCONE) 2)
 (2 (EXECUTE check_behavior
             $STOPPED
             $vehicle_stopped) 3)
 (3 (FACT vehicle_stopped $value) 4)
 (OR
   ((4 (TEST (== $value "False")) LOOP 2))
   ((4 (TEST (== $value "True")) 5)
    (5 (EXECUTE check_behavior
              $REACHEDCONE
              $cone_reached) 6)
    (6 (ASSERT  cone_reached
              $cone_reached) 7)))

/*
 start approach cone behavior
 while (not done)
   if (vehicle stopped)
      done = true
   if (reached cone)
      assert at cone
*/
}

//-------------------------------------
// KA 6
//-------------------------------------
KA {
NAME:
 "traveled_off_road"

DOCUMENTATION:
 "When the vehicle is at the cone,
  it does some off roading"

PURPOSE:
 (ACHIEVE traveled_off_road)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT vehicle_reached "False")
 (FACT STOPPED $STOPPED)
 (FACT REACHEDVEHICLE $REACHEDVEHICLE)
 (FACT OFFROAD $OFFROAD)

BODY:
 (1 (EXECUTE start_behavior $OFFROAD) 2)
```

```
 (2 (EXECUTE check_behavior
            $STOPPED $vehicle_stopped) 3)
 (3 (FACT vehicle_stopped $value) 4)
 (OR
   ((4 (TEST (== $value "False")) LOOP 2))
   ((4 (TEST (== $value "True")) 5)
    (5 (EXECUTE check_behavior
                $REACHEDVEHICLE
                $vehicle_reached) 6)
    (6 (ASSERT  vehicle_reached
                $vehicle_reached) 7)))

/*
   start off road behavior
   while (not done)
     if (vehicle stopped)
           done = true
   if (reached vehicle)
     assert at vehicle
*/
}

//-----------------------------------
// KA 7
//-----------------------------------
KA {
NAME:
 "vehicle_status_checked"

DOCUMENTATION:
 "Check to make sure the vehicle is ok
  once in a while"

PURPOSE:
 (ACHIEVE vehicle_status_checked)

CONTEXT:
 (FACT CHECKVEHICLE $CHECKVEHICLE)
 (FACT VEHICLESTATUS $VEHICLESTATUS)

BODY:
 (1 (EXECUTE start_behavior
             $CHECKVEHICLE) 2)
 (2 (EXECUTE check_behavior
             $VEHICLESTATUS
             $vehicle_status) 3)
 (3 (ASSERT vehicle_status
            $vehicle_status) 4)

 /*
    if (vehicle status != OK)
      stop vehicle, stop prs
 */
 }
```

References

[1] Eugene Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, Winter 1991.

[2] Department of the Army, Washington, D.C. *Tank Platoon, FM 17-15*, October 1987.

[3] Michael P. Georgeff and François Félix Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 972–978, Detroit, Michigan, 1989.

[4] Michael P. Georgeff and Amy L. Lansky. Procedural knowledge. *Proceedings of the IEEE*, 74(10):1383–1398, October 1986.

[5] François Félix Ingrand and Michael P. Georgeff. Managing deliberation and reasoning in real-time AI systems. In *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 284–291, San Diego, CA, November 1990.

[6] Francois F. Ingrand, Michael P. Georgeff, and Anand S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):34–44, December 1992.

[7] David Kinny, Magnus Ljungberg, and Anand Rao. Planned team activity. In Amedeo Cesta, Rosaria Conte, and Maria Miceli, editors, *Pre-Proceedings of the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, pages 1–20, Rome, Italy, July 1992.

[8] Charles Thorpe, Martial Hebert, Takeo Kanade, and Steven Shafer. Toward autonomous driving: The CMU navlab. *IEEE Expert*, pages 31–52, August 1991.